REMARKS

Claims 1 – 2 are pending in the application.

Claims 1 and 2 have been rejected under 35 USC §102 as being allegedly anticipated by Ji (US Patent No. 6,272,641).

In making the rejection, the Examiner contends that Ji. teaches the method of detecting malicious scripts as recited in claims 1 and 2 of the present invention. In particular, as per claim 1, the Examiner contends that Ji teaches –

> **A method of detecting malicious scripts using code insertion technique, comprising the step of: checking values related to each sentence belonging to call sequences by using method call sequence detection based on rules including matching rules and relation rules.....**

The Examiner relied on Col. 4, par. 5 of Ji, which recites –

A security policy defines what functions an applet needs to perform to be considered a security risk. Examples of security policies include preventing (1) applets from any file access, or (2) file access in a certain directory, or (3) creating certain Java objects. An applet scanner in accordance with the invention may allow different security policies for different clients, for different users, and for applets from different origins.

Col. 4, par. 5 of Ji fails to teach the afore-mentioned portion of claim 1 cited by the Examiner. Matching rules and relational rules are described in the above-identified application at paragraph 34 -

[par. 34] The matching rule takes the format of a general script sentence and further includes a rule variable (variable_string). In addition, the relation rule is composed of a condition phrase (condition_phrase) and action phrase

(action_phrase) and executes the content in the action phrase when the conditions of the condition phrase are satisfied. The condition phrase is composed of one or more condition expressions, each of which is described to check whether a specific rule has been already satisfied specific variable values of two rules are equal to each other, or one of the specific variable values is included in the other value.

Examples of matching rules and relation rules are further illustrated in the specification, by way of example, in FIGS. 1 and 2. In particular, FIG. 2 illustrates 7 matching rules, M1-M7, applicable to a Visual Basic Script code that performs self-replication via electronic mail (see Fig. 1). Similarly, FIG. 2 illustrates three relational rules, R1 – R3.

In stark contrast to the matching rules and relation rules of the invention, Ji at Col. 4, par. 5 is merely describing security policies which define what functions an applet needs to perform to be considered a security risk. This is clearly of an entirely different subject matter than claimed in claim 1. In otherwords, defining a general set of security policies, is clearly different subject matter and intent from composing matching rules comprised of script sentences which include rule variables and relational rules comprised of condition phrases and action phrases.

To further emphasize the difference between Ji and the recitation of claim 1, paragraph 40 of the specification describes the fine particularly exhibited by the matching and relation rules of the invention–

[par. 40] Symbol "*" used in the matching rules of the rule descriptions shown in FIG. 7 mean wildcards that can match with any kinds of tokens. In addition, the relation rules for checking only the presence of rules in the condition phrases operate by recognizing the right side of the action phrase as rule variables of rules satisfying the conditions even though additional rule IDs are not

described in the right side of the action phrase. For example, in the case of R4, the action phrase is interpreted as "$1=M2.$1" if the condition expression is satisfied because a rule of M2 is satisfied, and the action phrase is interpreted as "$1=R6.$1" if the condition expression is satisfied because a rule of R6 is satisfied. Therefore, the rule descriptions can be simplified.

II. In the Office Action, the Examiner further contends, with respect to claim 1, Ji teaches –

wherein the checking step comprises the steps of: inserting a self-detection routine (malicious behavior detection routine) call sentence before and after a method call sentence or an original script

The Examiner relies on Col. 3, par. 3 of Ji, which recites –

Thereby in accordance with the invention a scanner (for a virus or other malicious code) provides both static and dynamic scanning for application programs, e.g. Java applets or ActiveX controls. The applets or controls (hereinafter collectively referred to as applets) are conventionally received from e.g. the Internet or an Intranet at a conventional server. At this point the applets are statically scanned at the server by the scanner looking for particular instructions which may be problematic in a security context. The identified problematic instructions are then each instrumented, e.g. special code is inserted before and after each problematic instruction, where the special code calls respectively a prefilter and a post filter. Alternatively, the instrumentation involves replacing the problematic instruction with another instruction which calls a supplied function.

Ji describes the purpose of the pre and post filter calls at Col. 5, par. 3 –

An example of such a suspicious Java function is "Java.IO.File.list" which may list the contents of a client (local) directory 30, e.g. a directory on the client machine 14 hard disk drive. The first instruction sequence generates a call to a pre-filter function provided by the scanner 26, signaling that an insecure (suspicious) function is to be invoked. The pre-filter checks the security policy associated with the scanner 26 and decides whether this particular instruction ("call") is allowed. The second instruction sequence generates a call to a post-filter function also provided by the scanner. It also reports the result of the call to the post-filter function. Both the pre- and post-filter functions update the session state to be used by the security policy. The static scanning and instrumentation are both performed on the HTTP proxy server 32.

Col. 3, par. 3 of Ji fails to teach the afore-mentioned portion of claim 1 cited by the Examiner. The self detection (malicious behavior) routine call sentence is well-defined by claim 2:

2. The method according to claim 1, wherein the self-detection routine **call sentence is composed of sentences for storing parameters and return values and calling a detection engine**, said sentences being inserted before and after the method call sentence when the method call sentence matches with contents described in the matching rule, and wherein the self-detection routine includes a rule-based detection engine for executing the relation rule related to a relevant matching rule when a method corresponding to the matching rule is called and detecting the presence of malicious behavior of the method call sequence, and methods for causing the parameters and return values of the method call sentence satisfying the matching rule to be stored into a buffer usable by the detection engine.

As recited in claim 2, the self detection (malicious behavior) routine call sentence includes a rule-based detection engine for executing a relation rule related to a relevant matching rule when a method corresponding to the matching rule is called. This is clearly different from Ji in which a pre-filter checks a security policy associated with a scanner 26 and decides whether the particular instruction ("call") is allowed.

III. In the Office Action, the Examiner further contends, with respect to claim 1, Ji teaches –

Detecting the malicious codes during execution of the script through a self-detection routine inserted into the original script.

The Examiner relies on Col. 3, par. 4 of Ji, which recites –

The instrumented applet is then downloaded from the server to the client (local computer), at which time the applet code is conventionally interpreted by the client web browser and it begins to be executed. As the applet code is executed, each instrumented instruction is monitored by the web browser using a monitor package which is part of the scanner and delivered to the client side. Upon execution, each instrumented instruction is subject to a security check. If the security policy (which has been pre-established) is violated, that particular instruction which violates the security policy is not executed, and instead a report is made and execution continues, if appropriate, with the next instruction.

Col. 3, par.4 of Ji fails to teach the afore-mentioned portion of claim 1 cited by the Examiner.

The monitor package of Ji contains monitoring functions that are delivered from the server 32 to the client web browser 22 with the instrumental applet and are invoked by the instrumentation code in the applet. The monitor package includes a security policy checker which performs a security check to determine if a security policy is violated. If it is determined that a security policy has been violated, the particular instruction which violates the security policy is not executed, and instead a report is made and execution continues, if appropriate, with the next instruction.

In contrast to Ji, the self-detection routine of the invention is generated by a script transformer 20 (see Fig. 4) which transforms an original script 2 (see Fig. 4) including method call sentences into a script 6 (see Fig. 4) capable of continuously performing the self-detection during the execution through the method call sequence based on the detection rules 4 and the self-detection routine generated in the self-detection routine generator 10.

By transforming an original script 2, any malicious script entering from the outside or suspected of its maliciousness is transformed so that it can be self-detected continuously at a predetermined time before the execution. The script transformer 20 does not modify a sentence itself described in the original script 2 but merely causes additional codes to be inserted into the script.

It is respectfully submitted that the self-detection routine of the invention is functionally distinct from the monitor package of Ji for determining whether a security policy is violated. In particular, the self-detection routine of the invention is configured to include –

(1) a rule-based detection engine for executing the relation rule related to a relevant matching rule when a method corresponding to the matching rule is called,

(2) means for detecting the presence of malicious behavior of the method call sequence, and

(3) methods for causing the parameters and return values of the method call sentence satisfying the matching rule to be stored into a buffer usable by the detection engine.

Accordingly, it is believed that Applicants' Claim 1 recites patentable subject matter, and therefore, withdrawal of the rejections with respect to Claim 1 and allowance thereof is respectfully requested.

IV. In the Office Action, the Examiner has rejected Claim 2 under 35 U.S.C. §102.

Claim 2 depends from Claim 1 and therefore include the limitations of Claim 1. Accordingly, for the same reasons given above for Claim 1, Claim 2 is believed to contain patentable subject matter.

Additionally it is further asserted Claim 2 is patentable for at least the following reasons.

Fig. 2 of Ji, as cited by the Examiner, illustrates a monitor package containing monitoring functions that are delivered from the server 32 to the client web browser 22 with the instrumental applet and are invoked by the instrumentation code in the applet. The monitor package also creates a unique session upon instantiation. It also contains a security policy checker (supplied by security policy generator component 54) to determine whether the applet being

scanned violates the security policy, given the monitoring information. The security policy generator component 54 generates the security checker code included in the monitor package, from a set of predefined security policies. Different clients, users, and applets may have different security policies. The security policy generator 54 may run on server machine 20 or another computer. In addition, security policies can be configured by an administrator of the system. A simple security policy is to assign different weights to monitored functions and make sure the security weight of a session does not exceed a preset threshold. A more sophisticated security policy checks the file or resource the applet is trying to access at run time and prompts the user whether to allow the access. Hence the security policy broadly is a state machine to detect security policy violations upon attempted instruction execution.

As stated above, Claim 2 is directed to a self-detection routine call sentence comprised of (1) sentences for storing parameters and return values, and (2) sentences for calling a detection engine. (3) a rule based detection engine for executing relational rules related to a relevant matching rule when a method corresponding to a matching rule is called, (4) methods for detecting the presence of malicious behavior of the method call sequence, and (5) methods for causing the parameters and return values of the method call sentence satisfying the matching rule to be stored into a buffer usable by the detection engine. These elements are clearly distinct from the monitor package of Ji.

Accordingly, withdrawal of the rejections with respect to Claim 2 and allowance thereof are respectfully requested.

Therefore, in view of the forgoing remarks, it is respectfully submitted all claims pending herein are in condition for allowance. Please contact the undersigned attorney should there be any questions. A petition for an automatic three-month extension of time for response under 37 C.F.R. §1.136(a) is enclosed in triplicate, together with the requisite petition fee and fee for the additional claims introduced herein.

Early favorable action is earnestly solicited.

Respectfully submitted,

George M. Kaplan
Registration No.: 28,375
Attorney for Applicants

DILWORTH & BARRESE, LLP
333 Earle Ovington Boulevard
Uniondale, New York 11553
Tel. No. (516) 228-8484
Fax No. (516) 228-8516